APEX 新加坡亚太交易所

APEX Trading System
MdUserAPI Interface Specifications

Version 1.01

Contents

File Version	Modified Date	Notes
V1.00	19th April 2017	Version 1.00
V1.01	15th Jan 2019	1. Updated function prototype, struct, ErrorID and ErrorMsg based on ApexMdAPI Win32 v1.2.6 L3K. 2. Improved formatting. 3. Changed logo in header and added page number in footer.

1. Introduction

1.1 MduserAPI Overview

MduserAPI is also a C++ based class library. Application program can use and extend the interface provided by the class library to implement all market data subscription and receiving functions. This class library includes the following five files:

File Name	File Description
ApexFtdcMduserApi.h	Header File for Market Data Interface
ApexFtdcUserApiStruct.h	Defines a series of business-related data structure header files
ApexFtdcUserApiDataType.h	Defines a series of data type header files required by the API
apexmduserapi.dll	Dynamic-link library(DLL) binary file
apexmduserapi.lib	Import library(.Lib) file
libapexmduserapi.so	Linux dynamic library

Windows API version supports MS VC 6.0 and MS VC.NET 2003 compiler and requires multi-threading compilation option/MT. Liunx API version is based on Redhat 6.3, gcc 4.4.6 and depends on OpenSSL library.

1.2 Supported Platforms of MduserAPI

- **Intel X86/WindowsXP**: including **.h** files, **.dll** files and **.lib** files
- **Linux RedHat6.3**: including **.h** files and **.so** files

2 Architecture

MduserAPI communicates with market data gateway of APEX Trading System via FTD Protocol that is based on TCP Protocol. Market data gateway is responsible for processing market data subscription query from Member System, as well as pushing the subscribed market data back to Member System.

2.1 Communication Mode

FTD involves the following three communication modes:

- Dialog Communication Mode
- Private Communication Mode
- Broadcast Communication Mode

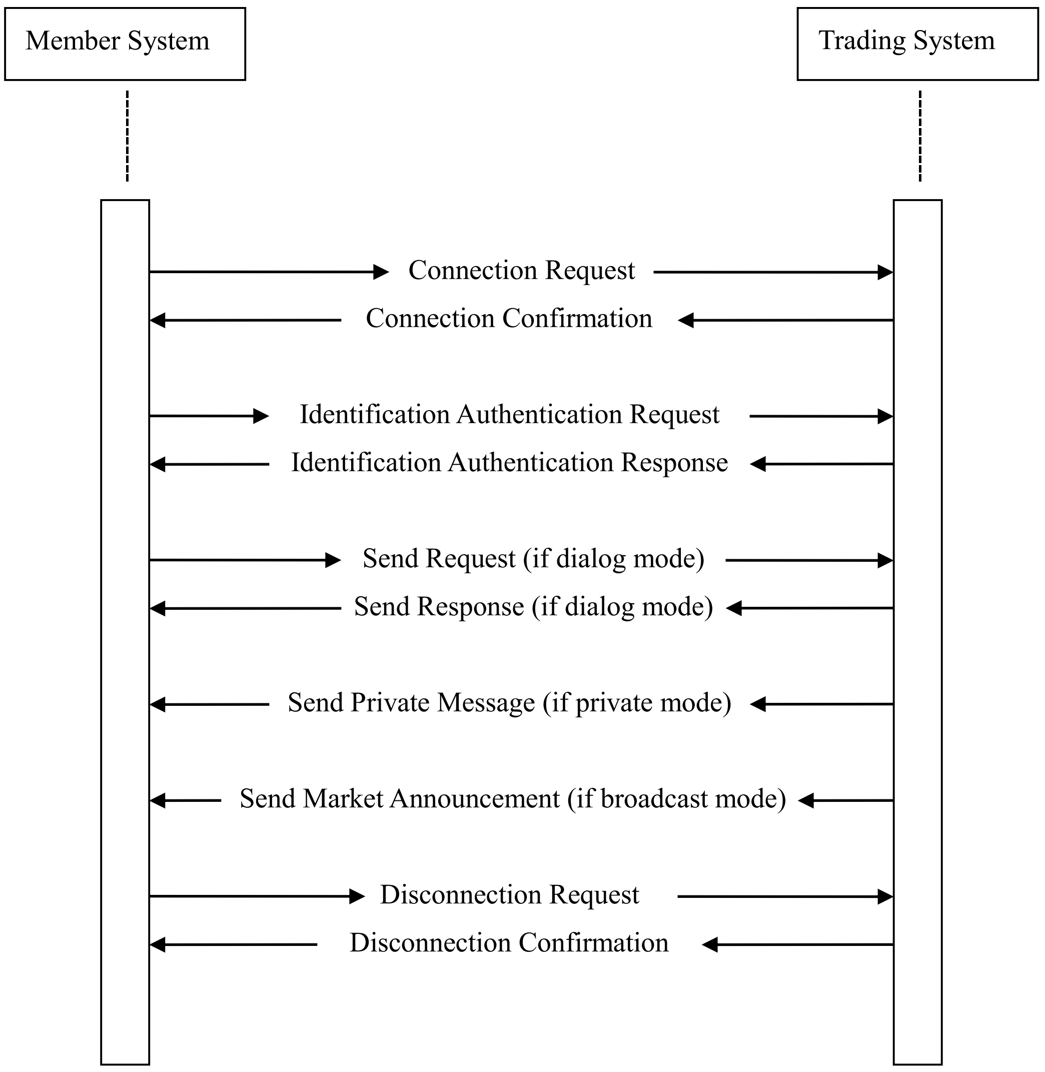
Dialog Communication Mode refers to communication whereby requests are initiated by Member System. Such requests (e.g. order, query, etc.) are received and processed by the Trading System and responses are sent back to the Member System. This is similar to the usual client/server mode.

Private Communication Mode means that trading system sends information (e.g. trade return) on its own initiative to a particular member or particular trader of particular member.

Broadcast Communication Mode means that trading system sends the same information (e.g. bulletin, public information in market etc.) to all traders.

Network connection and communication mode do not necessarily represent a simple one-to-one relationship. That is to say, message of different communication modes can be sent within one network connection, while messages of one communication mode can also be delivered within different connections.

In any of the communication modes, the communication process is the same, as depicted below:


Member System
Trading System
Connection Request
Connection Confirmation
Identification Authentication Request
Identification Authentication Response
Send Request (if dialog mode)
Send Response (if dialog mode)
Send Private Message (if private mode)
Send Market Announcement (if broadcast mode)
Disconnection Request
Disconnection Confirmation

2.2 Data Stream

Trading gateway supports dialog communication mode, private communication mode, and broadcast communication mode. The market data distribution function of market data gateway supports dialog communication mode and broadcast communication mode.

1) Dialog Communication Mode

Dialog Communication Mode supports **dialog data stream** and **query data stream**.

Dialog data stream is a bidirectional data flow through which Member System sends trading request and the trading system feeds back response. Trading system does not maintain the status of the dialog stream. In the event of a system failure, dialog data stream is reset and the data in transit might be lost.

Query data stream is also a bidirectional data flow through which member system sends query request and the trading system feeds back response. Trading system does not maintain the status of the query stream. In the event of a system failure, query data stream is reset and the data in transit might be lost.

2) Private Communication Mode

In the case of Private Communication Mode, data steam is reliable. Within a trading day, when Member System resumes its connection after a disconnection, Member System can request the trading system to resend the data within private data stream by specifying a starting sequence number. Private data stream provides Member System with order status report, trade return message and etc. Private data stream is classified into **member's private stream** and **trader's private stream**.

Trading system maintains a private data stream for each member. All return messages for a particular member such as order return and trade return, will be released through member's private stream. Only authorized traders can subscribe member's private stream.

Trader's private stream is similar to member's private stream, but it only covers return message for trades initiated by the trader himself. Every trader has the right to subscribe to his or her own trader's private stream.

3) Broadcast Communication Mode

Broadcast Communication Mode supports public data stream.

Public data stream is a uni-directional data stream that is sent from trading system or market data system to Member System for delivering public market information. Public data stream is a reliable data stream. Trading System maintains all public data streams within the system. Within a trading day, when Member System resumes its connection after a disconnection, Member System can request the trading system to resend the data within public data stream by specifying a starting sequence number.

Take market data as an example. Market data stream is a public data stream that is sent from Trading System to Member System for delivering market data information. Market data stream is a reliable data stream. Trading System maintains all the market data streams. Within a trading day, when Member System resumes its connection after a disconnection, Member System can request the trading system to resend the data within public data stream by specifying a starting sequence number.

Market data provided by the Trading System is organized according to topics. Each topic covers market data for a particular group of contracts, as well as market data release contents and release methods, including market depth, sample frequency, delay time etc. The Exchange announces the specific contents of each topic of market data and the topics of market data that can be subscribed by each market data user. Each market data topic corresponds with one market data stream.

In order to get market data, Member System must subscribe to one or more market data release topics after connecting with the gateway.

3 Interfaces

3.1 MduserAPI Interface

MduserAPI provides two interfaces, namely **CApexFtdcMduserApi** and **CApexFtdcMduserSpi**. These two interfaces encapsulate the FTD Protocol.

Member System can send operation requests via **CApexFtdcMduserApi** and it can process the response and reply from the APEX Trading System by inheriting **CApexFtdcMduserSpi** and overriding the callback functions.

3.1.1 Dialog Stream Programming Interface

The programming interface for communication through dialog stream typically looks like below.

```
   ////Request:
int CApexFtdcMduserApi::ReqXXX(CApexFtdcXXXField *pReqXXX,
                          int nRequestID)
   ////Response:
void CApexFtdcMduserSpi::OnRspXXX(CApexFtdcXXXField *pRspXXX,
                    CApexFtdcRspInfoField *pRspInfo,
                    int nRequestID,
                    bool bIsLast)
```

The 1st parameter for the request interface is the requested content, and it cannot be left empty. The 2nd parameter of the request interface is the request ID. The request ID is maintained by Member System and every request ID should be unique. The request ID filled in upon sending the request will be sent to Member System together with the response from the APEX Trading System, and Member System can match a particular request with its corresponding response by using this number.

The **CApexFtdcMduserSpi** callback function is called upon getting reply from the Trading System. If there are more than one piece of response data, the callback function/method will be called multiple times.

The callback function requires 4 input parameters:

- The 1st parameter is the actual data in the response. If there is an error in the process or if there is no such result, this field may be NULL.
- The 2nd parameter is the response info, indicating whether the current request is a success or a failure. If multiple callbacks occur, the value for this parameter from the 2nd callback onwards might all be NULL.
- The 3rd parameter is the request ID filled in when sending the request.
- The 4th parameter is the flag for the end of response, indicating whether this is the last callback for the current response.

3.1.2 Public Stream Programming Interface

Public stream carries market data information released by the Trading System.

The programming interface for communication through public stream typically looks like below.

```
void CApexFtdcMduserSpi::OnRtnXXX(CApexFtdcXXXField *pXXX);
```

The **CApexFtdcMduserSpi** callback function will be called upon receiving market data from the Trading System via the public stream. The parameter of the callback function is the content of the message.

4 Operating Mode

4.1 Workflow

The interaction between Member System and the APEX Trading System can be divided into two stages: the initialization phase and the function calling phase.

4.1.1 Initialization Phase

In the initialization phase, Member System has to complete the steps below (for more details, please refer to the codes in the **Development Example** section).

Steps	**Member System**
1	Generate an instance of **CApexFtdcMduserApi**
2	Generate an event handler instance
3	Register an event handler instance
4	Subscribe to the market data stream
5	Set the network address for the market data **NameServer** [1]
6	Initialization

[1]In order to be compatible with the previous version, this API still provides interfaces for the registration of the trading gateway (and market data gateway). However, APEX recommends not using these interfaces directly, which will be removed in the next version. Please refer to Section **4.5 Gateway List** for more details of the **NameServer**.

4.1.2 Function Calling Phase

In the function calling phase, Member System can call any of request methods from the **MduserApi** interface, e.g. ReqUserLogin, ReqSubscribeTopic, etc, and also provide callback functions to respond to return messages. It should be noted that:

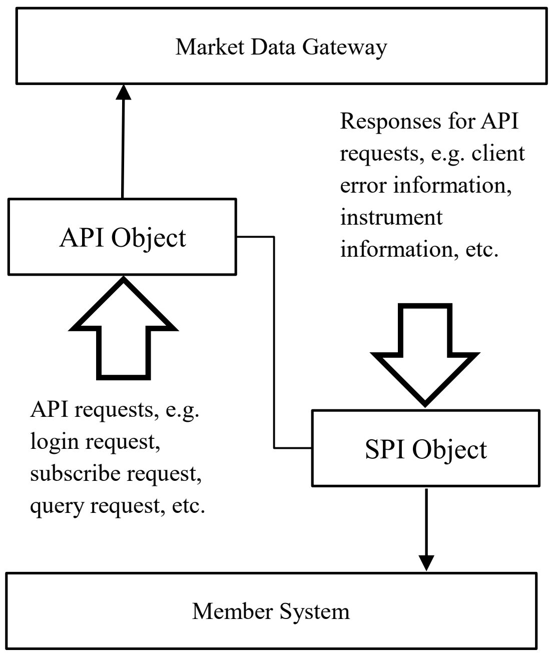
1) Input parameters for the API request function cannot be NULL.
2) The meaning of the output parameter returned from the API request function is: 0 stands for success, other numbers indicate an error.

4.2 Working Thread

The application program of Member System consists of at least two threads: one is the application program as the main thread, and the other is the API working thread. The communication between the application program and the market data gateway is driven by the API working thread.

The interface callback provided by CApexFtdcMduserSpi is driven by the MduserAPI working thread. It collects the required data from market data gateway by implementing the interface method of SPI.

If the callback function of the Member System application program blocks, MduserAPI working thread will also be blocked. In this case, the communication between API and market data gateway will stop, therefore quick return is required for callback functions. In the callback functions of derived classes of CApexFtdcMduserSpi, the quick return can be achieved by storing the data into the buffer or via the Windows messaging mechanism.


Market Data Gateway
Responses for API requests, e.g. client error information, instrument information, etc.
API Object
API requests, e.g. login request, subscribe request, query request, etc.
SPI Object
Member System

4.3 Connection to the gateway of the Trading System

MduserAPI communicates with market data gateways of APEX via the FTD Protocol which is built upon the TCP Protocol. **MduserAPI** uses the **CApexFtdcMduserApi::RegisterFront** to register the network address of the market data gateway.

APEX owns multiple trading and market data gateways, for both load balancing and backup purposes, to improve system performance and reliability. In order to guarantee the reliability for communications during trading hours, MduserAPI may register multiple gateways. After the API is initialized, it will randomly choose one gateway from the registered gateways and try to establish network connection with it. If the attempt fails, it will try other registered gateways one by one until the connection is successful. If there is network failure during trading process, the API will attempt to connect to the other gateways in a similar way.

APEX announces network addresses for at least 2 gateways. Hence, the Member System should register at least 2 gateway network addresses to prevent single point of failure resulting from the failure of the connected gateway.

APEX will use NameServer and will only publish NameServer addresses but not gateway addresses. MdUserAPI uses the **CApexFtdcMduserApi::RegisterNameServer** method to register the network address of APEX NameServer. the method can be called multiple times to register multiple addresses.

4.4 Local Files

During runtime, **MduserAPI** writes some data into local files. When calling the **CreateFtdcMduserApi** function, an input parameter can be passed to specify the local file path. This path must be created before runtime. The file extension of all local files is "**.con**". Different users should specify different local file path, otherwise they may not be able to receive some data from the Trading System.

4.5 Gateway List

For fault tolerance and load balancing, APEX deploys two groups of gateways at both the main data center and the backup data center. APEX publishes a list of gateway network addresses. The Member System can randomly choose a gateway from the list to attempt to establish connection with it. The Member System can only connect to one gateway at a certain moment. If the connected gateway encounters a problem and results in connection failure or timeout, the Member System should try the other gateways in the list.

There are two ways for Member System to obtain the gateway list:

1) APEX announces the gateway list. The Member System registers the front-ends of the list one by one into the API via the **RegisterFront** interface of API.
2) The Trading System provides **NameServer** to publish the gateway list for the API. APEX firstly announces the **NameServer** list, then the Member System registers the NameServer list into the API via the **RegisterNameServer** interface. The API first attempts to obtain the gateway list from the **NameServer**, then it will connect to one gateway based on the gateway list.

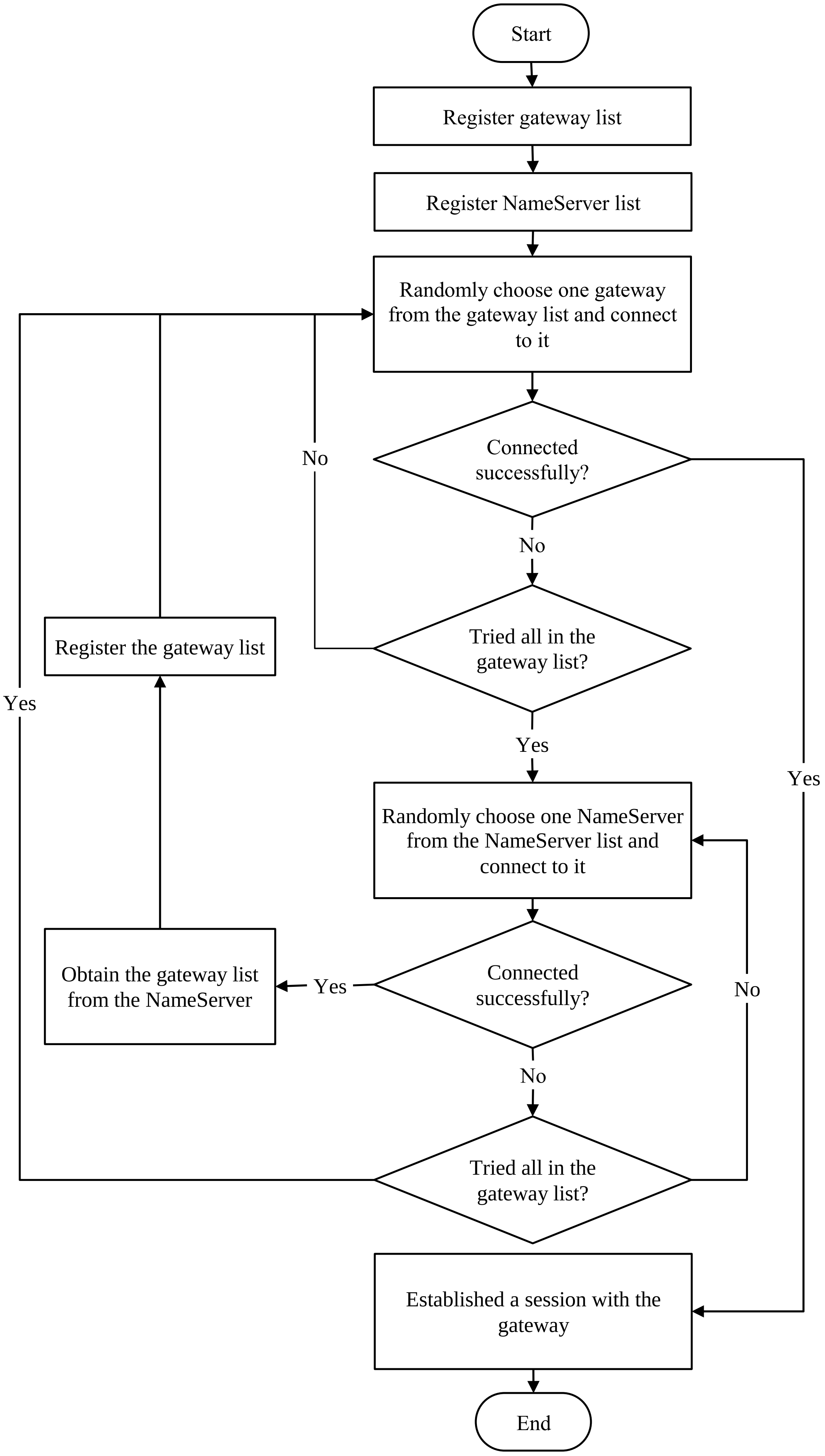
Advantages of employing NameServer include:

- APEX has more flexibility in gateway deployment. It can deploy additional gateways within a short time according to business requirement and load, without making any modification to the Member System.
- **NameServer** provides a better way for switching between the main system and disaster recovery system

- **NameServer** is characterized by its unique function, simple structure and low load. There is no need to worry about load balancing. Hence, it can be deployed in a flexible way.

The Member System can simultaneously use the **RegisterFront()** method to register the gateway list, and use the **RegisterNameServer()** method to register the **NameServer** list. API will first attempt to connect to its existing registered gateway. If unsuccessful, it will try to connect the NameServer.

The flow chart for the API to connect to the gateway is as below:


Start
Register gateway list
Register NameServer list
Randomly choose one gateway from the gateway list and connect to it
Connected successfully?
No
Yes
Tried all in the gateway list?
No
Yes
Register the gateway list
Randomly choose one NameServer from the NameServer list and connect to it
Connected successfully?
Yes
Obtain the gateway list from the NameServer
No
Tried all in the gateway list?
No
Yes
Established a session with the gateway
End

5 Categories of MduserAPI Interfaces

5.1 Management Interfaces

MduserAPI management interfaces control the life cycle and operating parameter of the API.

Interface Type	Interface name	Explanation
Lifecycle Management Interfaces	**CApexFtdcMduserApi::CreateFtdcMduserApi**	Create an **MduserApi** instance
	CApexFtdcMduserApi::GetVersion	Get API version
	CApexFtdcMduserApi::Release	Delete the instance of the interface
	CApexFtdcMduserApi::Init	Initialization
	CApexFtdcMduserApi::Join	Wait for the Interface thread to end the run
Parameter Management Interfaces	**CApexFtdcMduserApi::RegisterSpi**	Register callback interface
	CApexFtdcMduserApi::RegisterFront	Register **gateway** network address
	CApexFtdcMduserApi::RegisterNameServer	Register **NameServer** Network address
	CApexFtdcMduserApi::SetHeartbeatTimeout	Set the heartbeat timeout
Subscription Interfaces	**CApexFtdcMduserApi::SubscribeMarketDataTopic**	Subscribe to market data
Communication Status Interfaces	**CApexFtdcMduserSpi::OnFrontConnected**	The method is called when communication with the Trading System connection (prior to login) is established.
	CApexFtdcMduserSpi::OnFrontDisconnected	This method is called when communication with the Trading System is disconnected.
	CApexFtdcMduserSpi::OnHeartBeatWarning	The method is called when no heartbeat message is received after a long time.

5.2 Service Interfaces

Service Type	Service	Request Interface / Response Interface	Data Stream
Login-Logout	Login	**CApexFtdcMduserApi::ReqUserLogin** **CApexFtdcMduserSpi::OnRspUserLogin**	N/A
	Logout	**CApexFtdcMduserApi::ReqUserLogout** **CApexFtdcMduserSpi::OnRspUserLogout**	Dialog Stream
Subscription	Topic/Theme Subscription	**CApexFtdcMduserApi::ReqSubscribeTopic** **CApexFtdcMduserSpi::OnRspSubscribeTopic**	Dialog Stream
	Query Subscription	**CApexFtdcMduserApi::ReqQryTopic** **CApexFtdcMduserSpi::OnRspQryTopic**	Query Stream
Query	Query Instrument	**CApexFtdcMduserApi::ReqQryInstrument** **CApexFtdcMduserSpi::OnRspQryInstrument**	Query Stream
	Query Instrument Status	**CApexFtdcMduserApi::ReqQryInstrumentStatus** **CApexFtdcMduserSpi::OnRspQryInstrumentStatus**	Query Stream
Market Data	Market Data Notification	**CApexFtdcMduserSpi::OnRtnDepthMarketData**	Public Stream

6 MduserAPI Reference Manual

6.1 CApexFtdcMduserSpi Interface

CApexFtdcMduserSpi implements event notification interface. Member System has to derive the

CApexFtdcMduserSpi interface and provide event-handling methods/functions to handle events of interest.

6.1.1 OnFrontConnected Method

After the TCP virtual link path connection between Member System and the APEX Trading System is established, the method is called.

Function Prototype:

```
    void OnFrontConnected();
```

Note: The fact that the **OnFrontConnected** is called only implies that the TCP connection is successful. Member System must login separately to carry out any operations afterwards. Login failure will not callback this method.

6.1.2 OnFrontDisconnected Method

After connection between Member System and the APEX Trading System is broken, the method is called. In this case, API will automatically reconnect, and Member System does not need to deal with the reconnection. The automatically reconnected address may be the originally registered address or other available communication addresses that are supported by the system, which is decided by the API.

Function Prototype:

```
    void OnFrontDisconnected(int nReason);
```

Parameter: nReason: disconnection reasons

- 0x1001 network reading failure
- 0x1002 network writing failure
- 0x2001 heartbeat receiving timeout
- 0x2002 heartbeat sending timeout
- 0x2003 error message received

6.1.3 OnHeartBeatWarning Method

This method is called if packets/message is not received after a long time.

Function Prototype:

```
    void OnHeartBeatWarning(int nTimeLapse);
```

Parameter: nTimeLapse: time lapse from last time receiving the message (in seconds)

6.1.4 OnPackageStart Method

This method indicates the start of message/packets callback. After the API receives message/packet, it first calls this method, followed by the callback of the various data fields and then it calls OnPackageEnd to indicate the end of message callback.

Function Prototype:

```
    void OnPackageStart (int nTopicID, int nSequenceNo);
```

Parameters:

nTopicID: Topic ID (e.g. private stream, public stream, market data stream etc.)

nSequenceNo: Message Sequence Number

6.1.5 OnPackageEnd Method

This method indicates the end of message/packets callback. After the API receives a message/packet, it first calls OnPackageStart to indicate the start of message/packet callback, followed by the callback of the various data fields and then it calls this method.

Function Prototype:

```
    void OnPackageEnd (int nTopicID, int nSequenceNo);
```

Parameters:

nTopicID: Topic ID (e.g. private stream, public stream, market data stream etc.)

nSequenceNo: Message Sequence Number

6.1.6 OnRspUserLogin Method

After Member System sends out login request, and the Trading System sends back the response, the Trading System calls this method to inform the Member System whether the login is successful.

Function Prototype:

```
    void OnRspUserLogin(
            CApexFtdcRspUserLoginField *pRspUserLogin,
            CApexFtdcRspInfoField *pRspInfo,
            int nRequestID,
            bool bIsLast);
```

Parameters:

pRspUserLogin: returns the address for user login information structure:

```
    struct CApexFtdcRspUserLoginField {
        ///Trading day
        TApexFtdcDateType   TradingDay;
        ///Successful log in time
        TApexFtdcTimeType   LoginTime;
        ///Maximum order's local ID, NOT USED
        TApexFtdcOrderLocalIDType  MaxOrderLocalID;
        /// Transaction user's code
        TApexFtdcUserIDType UserID;
        ///Member ID
        TApexFtdcParticipantIDType  ParticipantID;
        ///Trading System Name
        TApexFtdcTradingSystemNameType TradingSystemName;
        ///Data Center ID
        TApexFtdcDataCenterIDType  DataCenterID;
        ///Current size of member's private flow
        TApexFtdcSequenceNoType PrivateFlowSize;
        /// Current size of private flow of trader/user
        TApexFtdcSequenceNoType UserFlowSize;
    };
```

pRspInfo: returns the address for user response information. **Special attention**: **when there are continuous successful response data, some returned value in between may be NULL, but the 1st returned value will never be NULL. This is the same below.** Error ID 0 means successful operation. This is the same below. Response information/message structure is:

```
    struct CApexFtdcRspInfoField {
        /// Error code
        TApexFtdcErrorIDType    ErrorID;
        /// Error message
        TApexFtdcErrorMsgType   ErrorMsg;
    };
```

nRequestID: returns the user login request ID; this ID is specified by the user upon login

bIsLast: indicates whether current return is the last return with respect to the nRequestID

6.1.7 OnRspUserLogout Method

After Member System sends out logout request and the Trading System sends back the response, this method is called to inform the Member System whether logout is successful.

Function Prototype:

```
   void OnRspUserLogout(
       CApexFtdcRspUserLogoutField *pRspUserLogout,
        CApexFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspUserLogout: returns the address for user logout information/message.

User logout information/message structure:

```
    struct CApexFtdcRspUserLogoutField {
        ///Transaction user's code
        TApexFtdcUserIDType UserID;
        ///Member code
        TApexFtdcParticipantIDType  ParticipantID;
    };
```

pRspInfo: returns address for user response information/message. The structure:

```
    struct CApexFtdcRspInfoField {
        /// Error code
        TApexFtdcErrorIDType    ErrorID;
        ///Error message
        TApexFtdcErrorMsgType   ErrorMsg;
    };
```

nRequestID: returns user logout request ID; this ID is specified by the user upon logout

bIsLast: indicates whether current return is the last return with respect to the nRequestID

6.1.8 OnRspSubscribeTopic Method

After Member System sends out topic subscription instruction, the API calls this method to send back the response.

Function Prototype:

```
    void OnRspSubscribeTopic (
        CApexFtdcDisseminationField *pDissemination,
        CApexFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pDissemination: points to the address for subscription topic structure, including topic subscribed and starting message sequence number. Subscription topic structure is:

```
    struct CApexFtdcDisseminationField {
        ///Sequence series
        TApexFtdcSequenceSeriesType SequenceSeries;
        ///Sequence number
        TApexFtdcSequenceNoType SequenceNo;
    };
```

pRspInfo: points to the address for response information/message structure. The structure:

```
    struct CApexFtdcRspInfoField {
        ///Error code
        TApexFtdcErrorIDType    ErrorID;
        ///Error message
        TApexFtdcErrorMsgType   ErrorMsg;
    };

    The possible errors:
    Error code  Error message               Possible reasons
    1           Not login                   Not logged in yet
```

nRequestID: returns subscribed topic request ID; this ID is specified by user upon subscription

bIsLast: indicates whether current return is the last return with respect to the nRequestID

6.1.9 OnRspQryTopic Method

After Member System sends out topic query instruction, the API calls this method to send back the response.

Function Prototype:

```
    void OnRspQryTopic (
        CApexFtdcDisseminationField *pDissemination,
        CApexFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pDissemination: points to the address for topic query structure, including topic queried and number of messages in the topic. Topic query structure is:

```
    struct CApexFtdcDisseminationField {
        ///Sequence series
        TApexFtdcSequenceSeriesType SequenceSeries;
        ///Sequence number
        TApexFtdcSequenceNoType SequenceNo;
    };
```

pRspInfo: points to the address for response information/message structure. The response information/message structure is:

```
    struct CApexFtdcRspInfoField {
        ///Error code
        TApexFtdcErrorIDType    ErrorID;
        ///Error message
        TApexFtdcErrorMsgType   ErrorMsg;
    };

    The possible errors:
    Error code  Error message               Possible reasons
    1           Not login                   Not logged in yet
```

nRequestID: returns the topic query request ID, specified upon sending topic query request.

bIsLast: indicates whether current return is the last return with respect to the nRequestID.

6.1.10 OnRspQryInstrument Method

After Member System sends out instrument query request, the API calls this method to send back the response.

Function Prototype:

```
    void OnRspQryInstrument(
        CApexFtdcRspInstrumentField *pRspInstrument,
        CApexFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspInstrument: points to the address for instrument structure:

```
    struct CApexFtdcRspInstrumentField {
        ///Settlement Group ID
        TApexFtdcSettlementGroupIDType SettlementGroupID;
        ///Product ID
        TApexFtdcProductIDType ProductID;
        ///Product Group ID
        TApexFtdcProductGroupIDType ProductGroupID;
        ///Underlying Instrument ID
        TApexFtdcInstrumentIDType UnderlyingInstrID;
        ///Product Class
        TApexFtdcProductClassType ProductClass;
        ///Position Type
        TApexFtdcPositionTypeType PositionType;
        ///Strike Price
        TApexFtdcPriceType  StrikePrice;
        ///Options Type
        TApexFtdcOptionsTypeType OptionsType;
        ///Volume Multiple
        TApexFtdcVolumeMultipleType VolumeMultiple;
        ///Underlying Multiple
        TApexFtdcUnderlyingMultipleType UnderlyingMultiple;
        ///Instrument ID
        TApexFtdcInstrumentIDType InstrumentID;
        ///Instrument Name
        TApexFtdcInstrumentNameType InstrumentName;
        ///Delivery Year
        TApexFtdcYearType DeliveryYear;
        ///Delivery Month
        TApexFtdcMonthType DeliveryMonth;
        ///Advance Month
        TApexFtdcAdvanceMonthType AdvanceMonth;
        ///A flag indicating Is Trading now
        TApexFtdcBoolType IsTrading;
        ///Currency ID
        TApexFtdcCurrencyIDType CurrencyID;
        ///Create Date
        TApexFtdcDateType CreateDate;
        ///Open Date
        TApexFtdcDateType OpenDate;
        ///Expiry Date
        TApexFtdcDateType ExpireDate;
        ///Start Delivery Date
        TApexFtdcDateType StartDelivDate;
        ///End Delivery Date
        TApexFtdcDateType EndDelivDate;
        ///Basis Price
        TApexFtdcPriceType BasisPrice;
        ///Max Market Order Volume
        TApexFtdcVolumeType MaxMarketOrderVolume;
        ///Min Market Order Volume
        TApexFtdcVolumeType MinMarketOrderVolume;
        ///Max Limit Order Volume
        TApexFtdcVolumeType MaxLimitOrderVolume;
        ///Min Limit Order Volume
        TApexFtdcVolumeType MinLimitOrderVolume;
        ///Price Tick
        TApexFtdcPriceType PriceTick;
        ///Allow Deliver Person Open
        TApexFtdcMonthCountType AllowDelivPersonOpen;
    };
```

pRspInfo: points to the address for response information/message structure:

```
struct CApexFtdcRspInfoField {
    ///Error code
    TApexFtdcErrorIDType    ErrorID;
    ///Error message
    TApexFtdcErrorMsgType   ErrorMsg;
};

The possible errors:
Error code  Error message                Possible reasons
1           Not login                    Not logged in yet
```

nRequestID: returns the instrument query request ID, specified upon sending instrument query request
bIsLast: indicates whether current return is the last return with respect to the nRequestID.

6.1.11 OnRspQryInstrumentStatus Method

After Member System sends out instrument status query request, the API calls this method to send back the response.

Function Prototype:

```
    void OnRspQryInstrumentStatus(
        CApexFtdcInstrumentStatusField *pRspInstrumentStatus,
        CApexFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspInstrumentStatus: points to the address for instrument status structure:

```
    struct CApexFtdcInstrumentStatusField {
        ///Settlement Group ID
        TApexFtdcSettlementGroupIDType SettlementGroupID;
        ///Instrument ID
        TApexFtdcInstrumentIDType InstrumentID;
        ///Instrument Status
        TApexFtdcInstrumentStatusType InstrumentStatus;
        ///Trading Segment Sequence Number
        TApexFtdcTradingSegmentSNType TradingSegmentSN;
        ///Enter Time
        TApexFtdcTimeType EnterTime;
        ///Enter Reason
        TApexFtdcInstStatusEnterReasonType EnterReason;
        ///Calendar Date
        TApexFtdcDateType CalendarDate;
    };
```

pRspInfo: points to the address for response information/message structure:

```
    struct CApexFtdcRspInfoField {
        ///Error code
        TApexFtdcErrorIDType    ErrorID;
        ///Error message
        TApexFtdcErrorMsgType   ErrorMsg;
    };

    The possible errors:
    Error code  Error message               Possible reasons
    1           Not login                   Not logged in yet
```

nRequestID: returns the instrument status query request ID, specified upon sending instrument status query request

bIsLast: indicates whether current return is the last return with respect to the nRequestID.

6.1.12OnRtnDepthMarketData Method

Whenever there is any change to market data, the Trading System will inform Member System, and this method is called.

Function Prototype:

```
    void OnRtnDepthMarketData (CApexFtdcDepthMarketDataField *pTrade);
```

Parameter:

pDepthMarketData: points to the address for market data structure. Note: some fields in the market data are not used. The market data structure:

```
    struct CApexFtdcDepthMarketDataField {
        ///Business day
        TApexFtdcDateType  TradingDay;
        ///Settlement group's code
        TApexFtdcSettlementGroupIDType SettlementGroupID;
        ///Settlement No.
        TApexFtdcSettlementIDType  SettlementID;
        /// Latest price
        TApexFtdcPriceType  LastPrice;
        /// Yesterday's settlemnt
        TApexFtdcPriceType  PreSettlementPrice;
        ///Yesterday's close
        TApexFtdcPriceType  PreClosePrice;
        ///Yesterday's open interest
        TApexFtdcLargeVolumeType   PreOpenInterest;
        ///Today's open
        TApexFtdcPriceType  OpenPrice;
        ///The highest price
        TApexFtdcPriceType  HighestPrice;
        ///The lowest price
        TApexFtdcPriceType  LowestPrice;
        ///Quantity
        TApexFtdcVolumeType Volume;
        ///Turnover
        TApexFtdcMoneyType  Turnover;
        ///Open interest
        TApexFtdcLargeVolumeType   OpenInterest;
        ///Today's close
        TApexFtdcPriceType  ClosePrice;
        ///Today's settlement
        TApexFtdcPriceType  SettlementPrice;
        ///The upward price limit
        TApexFtdcPriceType  UpperLimitPrice;
        ///The downward price limit
        TApexFtdcPriceType  LowerLimitPrice;
        ///Yesterday's Delta value, not used
        TApexFtdcRatioType  PreDelta;
        ///Today's Delta value, not used
        TApexFtdcRatioType  CurrDelta;
        ///Last modification time
        TApexFtdcTimeType   UpdateTime;
        /// The last modified millisecond
        TApexFtdcMillisecType UpdateMillisec;
        ///Contract code
        TApexFtdcInstrumentIDType  InstrumentID;
        ///Bid price 1
        TApexFtdcPriceType  BidPrice1;
        /// Bid volume 1
        TApexFtdcVolumeType BidVolume1;
        ///Ask price 1
        TApexFtdcPriceType  AskPrice1;
        ///Ask volume 1
        TApexFtdcVolumeType AskVolume1;
        /// Bid price 2
        TApexFtdcPriceType  BidPrice2;
        /// Bid volume 2
        TApexFtdcVolumeType BidVolume2;
        /// Ask price 2
        TApexFtdcPriceType  AskPrice2;
        /// Ask volume 2
        TApexFtdcVolumeType AskVolume2;
        ///Bid price 3
        TApexFtdcPriceType  BidPrice3;
        ///Bid volume 3
        TApexFtdcVolumeType BidVolume3;
        /// Ask price 3
        TApexFtdcPriceType  AskPrice3;
        ///Ask volume 3
        TApexFtdcVolumeType AskVolume3;
        ///Bid price 4
        TApexFtdcPriceType  BidPrice4;
        ///Bid volume 4
        TApexFtdcVolumeType BidVolume4;
        ///Ask price 4
        TApexFtdcPriceType  AskPrice4;
        ///Ask volume 4
        TApexFtdcVolumeType AskVolume4;
        ///Bid price 5
        TApexFtdcPriceType  BidPrice5;
        ///Bid volume 5
        TApexFtdcVolumeType BidVolume5;
        /// Ask price 5
        TApexFtdcPriceType  AskPrice5;
        ///Ask price 5
        TApexFtdcVolumeType AskVolume5;
        ///upper price banding
        TApexFtdcPriceType  BandingUpperPrice;
        ///lower price banding
        TApexFtdcPriceType  BandingLowerPrice;
        ///Reference price
        TApexFtdcPriceType ReferencePrice;
        ///Calendar date
        TApexFtdcDateType CalendarDate;
    };
```

6.1.13OnRspError Method

This method is called when a request returns an error.

Function Prototype:

```
    void OnRspError(
        CApexFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspInfo: returns the address for response information/message structure. The response information/message structure is:

```
    struct CApexFtdcRspInfoField {
        ///Error code
        TApexFtdcErrorIDType    ErrorID;
        ///Error Message
        TApexFtdcErrorMsgType   ErrorMsg;
    };
```

nRequestID: returns the user operating request ID; this ID is specified upon sending request.

bIsLast: indicates whether current return is the last return with respect to the nRequestID.

6.2 CApexFtdcMduserApi Interfaces

Functions offered by **CApexFtdcMduserApi** interfaces include login/logout, market data subscription etc.

6.2.1 CreateFtdcMduserApi Method

Creates an instance of the **CApexFtdcMduserApi.** This cannot be created with a "new".

Function Prototype:

```
    static CApexFtdcMduserApi *CreateFtdcMduserApi(const char *pszFlowPath = "");
```

Parameter:

pszFlowPath: constant character pointer, used to point to a directory that stores the local files generated by the API. The default value is the current directory.

Return Value

This returns a pointer that point to an instance of the **CApexFtdcMduserApi**.

6.2.2 GetVersion Method

This is to get the API version.

Function Prototype:

```
    static const char *GetVersion();
```

Returned Value:

This returns a constant pointer that point to the versioning identification string.

6.2.3 Release Method

This is the proper method (instead of delete keyword) to release an instance of **CApexFtdcMduserApi.**

Function Prototype:

```
    void Release();
```

6.2.4 Init Method

Establishes the connection between Member System and the Trading System. After the connection is established, user can proceed to login.

Function Prototype:

```
    void Init();
```

6.2.5 Join Method

Member System waits for the end of an interface thread instance.

Function Prototype:

```
    void Join();
```

6.2.6 GetTradingDay Method

Gets the current trading day. The correct value can only be obtained after successful login to the Trading System.**Function Prototype:**

```
    const char *GetTradingDay();
```

Return Value:

This returns a constant pointer that point to the date information character string.

6.2.7 RegisterSpi Method

Registers an instance derived from **CApexFtdcMduserSpi** class that performs events handling.

Function Prototype:

```
    void RegisterSpi(CApexFtdcMduserSpi *pSpi);
```

Parameter:

pSpi: the pointer for **CApexFtdcMduserSpi** interface instance.

6.2.8 RegisterFront Method

Registers network communication address of Trading System gateway. The Trading System has multiple gateways and the Member System can register multiple network communication addresses of the gateways. This method has to be called before the **Init Method** is called.

Function Prototype:

```
    void RegisterFront(char *pszFrontAddress);
```

Parameter:

pszFrontAddress: a pointer that poits to the gateway network communication address. The server address is in the format "**protocol://ipaddress:port**", e.g. "tcp://127.0.0.1:17001". "tcp"in the example is the transmission protocol, "127.0.0.1" represents the server address, and "17001" represent s the server port number.

6.2.9 RegisterNameServer Method

Registers the network communication address of the Trading System NameServer. Since trading system has multiple NameServers, Member System can register multiple network communication addresses of NameServer. This method needs to be called before the Init method.

Function prototype:

```
    void RegisterNameServer (char *pszNsAddress);
```

Parameter:

pszNsAddress: Pointer pointing to network communication address of the Exchange's NameServer.

Format of network address: “protocol://ipaddress:port” such as “tcp://127.0.0.1:17001”. “tcp” represents the transmission protocol, “127.0.0.1” represents the server's address, and “17001” represents the port No. of server.

6.2.10 SetHeartbeatTimeout Method

Sets the heartbeat timeout limit for network communication. After the connection between MduserAPI and the Trading System is established, it will send regular heartbeat to detect whether the connection is functioning well. **The Exchange suggests members to set the timeout to be between 10s and 30s.**

Function Prototype:

```
    virtual void SetHeartbeatTimeout(unsigned int timeout);
```

Parameter:

Timeout: heartbeat timeout time limit (in seconds). If no information/message is received from the Trading System after “timeout/2” seconds, **CApexFtdcMduserApi::OnHeartBeatWarning()** will be called/triggered. If no information/message is received from the Trading System after “timeout” seconds, the connection will be stopped, and **CApexFtdcMduserApi::OnFrontDisconnected()** will be called/triggered

6.2.11 SubscribeMarketDataTopic Method

Subscribes market data topic specified by the Member System. After the subscription, the Trading System will proactively send out market data notification to Member System.

Function Prototype:

```
    void SubscribeMarketDataTopic(int nTopicID, APEX_TE_RESUME_TYPE nResumeType);
```

Parameters:

nTopicID: Market data topic to be subscribed, announced by the Exchange

nResumeType: Market data retransmission method type:

- TERT_RESTART: to retransmit from current trading day
- TERT_RESUME: to retransmit by resuming and continuing from last transmission
- TERT_QUICK: first transmit the market data snapshot, and then transmit all market data after that. **The Exchange recommends members to use this method to recover market data quickly.**

6.2.12 ReqUserLogin Method

This is the user login request.

Function Prototype:

```
    int ReqUserLogin(
        CApexFtdcReqUserLoginField *pReqUserLoginField,
        int nRequestID);
```

Parameters:

pReqUserLoginField: points to the address for login request structure:

```
    struct CApexFtdcReqUserLoginField {
        ///Trading Day
        TApexFtdcDateType   TradingDay;
        ///Trading User ID
        TApexFtdcUserIDType UserID;
        ///Member ID
        TApexFtdcParticipantIDType  ParticipantID;
        ///Password
        TApexFtdcPasswordType   Password;
        ///User-end product information
        TApexFtdcProductInfoType    UserProductInfo;
        ///Interface-port product information
        TApexFtdcProductInfoType    InterfaceProductInfo;
        ///Protocol information, NOT USED
        TApexFtdcProtocolInfoType   ProtocolInfo;
        ///Data Center ID
        TApexFtdcDataCenterIDType  DataCenterID;
    };
Member System needs to fill in the "UserProductInfo" and "InterfaceProductInfo" fields, i.e. product information of the Member System, such as software development vendor, version number etc., e.g. "ABC Market Data System V100"—the Member System developed by ABC firm and the version number.

If it is the first login, then "DataCenterID" should be 0 or the primary data center ID released by the Exchange. The DataCenterID value returned from previous login reply should be filled in for future login.
```

nRequestID: the request ID for login request; it is specified and managed by the user.

Returned Value:

- 0, represents successful
- -1, represents the network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity
- -3, indicates that the number of requests sent per second exceeds the allowable quantity

6.2.13 ReqUserLogout Method

This is the user logout request.

Function Prototype:

```
    int ReqUserLogout(
        CApexFtdcReqUserLogoutField *pReqUserLogout,
        int nRequestID);
```

Parameters:

pReqUserLogout: points to the address for logout request structure. The structure:

```
    struct CApexFtdcReqUserLogoutField {
        ///Trading User ID
        TApexFtdcUserIDType UserID;
        ///Member ID
        TApexFtdcParticipantIDType  ParticipantID;
    };
```

nRequestID: the request ID for logout request; it is specified and managed by the user.

Returned Value:

- 0, represents successful
- -1, represents the network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity
- -3, indicates that the number of requests sent per second exceeds the allowable quantity

6.2.14 ReqSubscribeTopic Method

This is the request to subscribe to topic/theme. This should be called after login.

Function Prototype:

```
    int ReqSubscribeTopic (
        CApexFtdcDisseminationField * pDissemination,
        int nRequestID);
```

Parameters:

pDissemination: points to the address for subscribed topic structure, including topic to be subscribed as well as the starting message sequence number. The structure:

```
    struct CApexFtdcDisseminationField {
        ///Sequence series number
        TApexFtdcSequenceSeriesType SequenceSeries;
        ///Sequence number
        TApexFtdcSequenceNoType SequenceNo;
    };
SequenceSeries: topics to be subscribed
SequenceNo: = -1 to retransmit using the "QUICK" method, = other value to resume transmission from
this sequence number onwards
```

nRequestID: the request ID; it is specified and managed by the user.

Returned Value:

- 0, represents successful
- -1, represents the network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity
- -3, indicates that the number of requests sent per second exceeds the allowable quantity

6.2.15 ReqQryTopic Method

This is the request for querying topic/theme. This should be called after login.

Function Prototype:

```
    int ReqQryTopic (
        CApexFtdcDisseminationField * pDissemination,
        int nRequestID);
```

Parameters:

pDissemination: points to the address for topic query structure, including topic to be queried. The structure:

```
    struct CApexFtdcDisseminationField {
        ///Sequence Series
        TApexFtdcSequenceSeriesType SequenceSeries;
        ///Sequence Number
        TApexFtdcSequenceNoType SequenceNo;
    };
    SequenceSeries: topics to be queried
    SequenceNo: no need to fill in
```

nRequestID: the request ID; it is specified and managed by the user.

Returned Value:

- 0, represents successful
- -1, represents the network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity
- -3, indicates that the number of requests sent per second exceeds the allowable quantity

6.2.16 ReqQryInstrument Method

This is the request for querying instruments. This should be called after login.

Function Prototype:

```
    int ReqQryInstrument(
        CApexFtdcQryInstrumentField *pQryInstrument,
        int nRequestID);
```

Parameters:

pQryInstrument: points to the address for instrument query structure:

```
    struct CApexFtdcQryInstrumentField {
        ///Settlement Group ID
        TApexFtdcSettlementGroupIDType SettlementGroupID;
        ///Product Group ID
        TApexFtdcProductGroupIDType ProductGroupID;
        ///Product ID
        TApexFtdcProductIDType ProductID;
        ///Instrument ID
        TApexFtdcInstrumentIDType InstrumentID;
    };
```

nRequestID: the request ID; it is specified and managed by the user.

Returned Value:

- 0, represents successful
- -1, represents the network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity
- -3, indicates that the number of requests sent per second exceeds the allowable quantity

6.2.17 ReqQryInstrumentStatus Method

This is the request for querying instrument statuses. This should be called after login.

Function Prototype:

```
    int ReqQryInstrumentStatus(
        CApexFtdcQryInstrumentStatusField *pQryInstrumentStatus,
        int nRequestID);
```

Parameters:

pQryInstrumentStatus: points to the address for instrument status query structure:

```
    struct CApexFtdcQryInstrumentStatusField {
        ///Instrument ID Start
        TApexFtdcInstrumentIDType InstIDStart;
        ///Instrument ID End
        TApexFtdcInstrumentIDType InstIDEnd;
    };
```

nRequestID: the request ID; it is specified and managed by the user.

Returned Value:

- 0, represents successful
- -1, represents the network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity
- -3, indicates that the number of requests sent per second exceeds the allowable quantity

7 MduserAPI—A Development Example

```
// A simple example to illustrate usage of CApexFtdcMduserApi &
CApexFtdcMduserSpi interfaces
#include <stdio.h>
#include <string.h>
#include "ApexFtdcMduserApi.h"

class CSimpleHandler : public CApexFtdcMduserSpi
{
public:

// constructor, requires a valid pointer that points to an instance of
CApexFtdcMduserApi
    CSimpleHandler(CApexFtdcMduserApi *pUserApi) : m_pUserApi(pUserApi) {}

    ~CSimpleHandler() {}

// After the communication connection between Member System and the Trading
System is established, Member System is required to log in.
    void OnFrontConnected() {
        CApexFtdcReqUserLoginField reqUserLogin;
        strcpy(reqUserLogin.ParticipantID, "P001");
        strcpy(reqUserLogin.UserID, "U001");
        strcpy(reqUserLogin.Password, "P001");

        m_pUserApi->ReqUserLogin(&reqUserLogin, 0);
    }

// when the communication connection between Member System and the Trading
System is interrupted, this method is called.
    void OnFrontDisconnected() {
// when disconnection happens, API will reconnect automatically, Member
System is not required to handle.
        printf("OnFrontDisconnected.\n");
    }

// after the Trading System sends out reply for login request, this method
is called to inform Member System whether the login is successful
    void OnRspUserLogin(CApexFtdcRspUserLoginField *pRspUserLogin,
CApexFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {
        printf("OnRspUserLogin: ErrorCode=[%d], ErrorMsg=[%s]\n",
     pRspInfo->ErrorID, pRspInfo->ErrorMsg);
        printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
        if (pRspInfo->ErrorID != 0) {
//login failure, Member System needs to do error handling
            printf("Failed to login, errorcode=%d errormsg=%s requestid=%d 
chain=%d", pRspInfo->ErrorID, pRspInfo->ErrorMsg, nRequestID, bIsLast);
        }
    }

//Depth market data notification, sent by the Trading System automatically.
    void OnRtnDepthMarketData(CApexFtdcDepthMarketDataField *pMarketData) {
//Member System should deal with the returned data based on its own need
    }

//error notification with respect to user request
    void OnRspError(CApexFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
{
        printf("OnRspError:\n");
        printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID, 
pRspInfo->ErrorMsg);
        printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
// Member System should do error handling
    }

private:
// a pointer that points to an instance of CApexFtdcMduserApi
    CApexFtdcMduserApi *m_pUserApi;
};

int main()
{
// creates an instance of CApexFtdcMduserApi
    CApexFtdcMduserApi *pUserApi = CApexFtdcMduserApi::CreateFtdcMduserApi();
// creates an instance for event handling
    CSimpleHandler sh(pUserApi);
// register an instance for event handling
    pUserApi->RegisterSpi(&sh);
// subscribes depth market data topic
/// TERT_RESTART: to retransmit from current trading day
/// TERT_RESUME: to retransmit by resuming and continuing from last 
transmission
/// TERT_QUICK: first transmit the market data snapshot, and then transmit all 
market data after that
    pUserApi-> SubscribeMarketDataTopic (102, TERT_RESUME);
    //set the timeout for heartbeat
    pUserApi->SetHeartbeatTimeout(19);
```

```
// set the Exchange NameServer addresses
    char *addresses[] = {
        "tcp://192.168.1.1:17011",
        "tcp://192.168.1.2:17011",
        "tcp://192.168.1.3:17011",
        "tcp://192.168.1.4:17011"
    };

    for (int i = 0; i < 4; i++) {
        pUserApi->RegisterNameServer(addresses[i]);
    }

// starts connection with market data gateway of the Trading System
    pUserApi->Init();
// release MduserAPI instance
    pUserApi->Release();
       return 0;
```